Knowledge and Information Systems (2024) 66:2393–2416
https://doi.org/10.1007/s10115-023-02024-z

REGULAR PAPER

A unified framework for backpropagation-free soft and hard gated graph neural networks

Luca Pasa[1] · **Nicolò Navarin**[1] · **Wolfgang Erb**[1] · **Alessandro Sperduti**[1,2]

Received: 1 March 2023 / Revised: 14 October 2023 / Accepted: 6 November 2023 /
Published online: 26 December 2023

Abstract
We propose a framework for the definition of neural models for graphs that do not rely on backpropagation for training, thus making learning more biologically plausible and amenable to parallel implementation. Our proposed framework is inspired by Gated Linear Networks and allows the adoption of multiple graph convolutions. Specifically, each neuron is defined as a *set* of graph convolution filters (weight vectors) and a gating mechanism that, given a node and its topological context, generates the weight vector to use for processing the node's attributes. Two different graph processing schemes are studied, i.e., a message-passing aggregation scheme where the gating mechanism is embedded directly into the graph convolution, and a multi-resolution one where neighboring nodes at different topological distances are jointly processed by a single graph convolution layer. We also compare the effectiveness of different alternatives for defining the context function of a node, i.e., based on hyperplanes or on prototypes, and using a soft or hard-gating mechanism. We propose a unified theoretical framework allowing us to theoretically characterize the proposed models' expressiveness. We experimentally evaluate our backpropagation-free graph convolutional neural models on commonly adopted node classification datasets and show competitive performances compared to the backpropagation-based counterparts.

Keywords Graph convolutional networks · Graph neural network · Deep learning · Structured data · Machine learning on graphs

This work was partly funded by the SID/BIRD project *Deep Graph Memory Networks*, Department of Mathematics, University of Padova. The authors acknowledge the HPC resources of the Department of Mathematics, University of Padova, made available for conducting the research reported in this paper.

✉ Luca Pasa
luca.pasa@unipd.it

1 Department of Mathematics, University of Padua, Padua, Italy

2 DISI, University of Trento, Trento, Italy

1 Introduction

In recent years, several definitions of neural architectures capable of dealing with data in structured forms, such as graphs, have been presented [1, 2]. The vast majority of graph neural networks in the literature are based on the idea of message passing [3], in which the representation of a node at layer l (or time t if the network is recurrent) is defined as a transformation of the attributes associated to the same node and the ones of its neighbors at layer $l-1$ ($t-1$).

While many works focused on defining alternative architectures, to the best of the authors' knowledge, all of them rely on backpropagation to learn the networks' weights. Backpropagation is a powerful and effective method to train deep neural networks (NNs). It has been successfully applied almost ubiquitously in recent years when the training of NNs was involved. However, when the amount of available data is not huge, the standard approach of training a nonlinear NN with backpropagation may quickly lead to overfitting the training data. This clearly contrasts how humans learn since we do not require nearly the amount of training data modern NNs do to learn how to generalize [4]. Moreover, the backpropagation mechanism is not biologically plausible [5, 6], suggesting that the brain may use different learning algorithms.

Some alternative definitions of multilayer neural networks that do not rely on backpropagation for their training [4, 6] have been proposed. They define local learning rules where each neuron, given its inputs, is trained independently from the rest of the network exploiting a global error signal. This approach allows these networks to: (i) be more biologically plausible (i.e., from the current knowledge about the functioning of animal neurons, it seems implausible for a neuron to have access to the connections in a brain area responsible for a subsequent processing step); (ii) be more sample efficient/simplify the overall training procedure since each neuron solves an independent (possibly convex) problem.

The aim of this paper is to explore the contamination between these two cutting-edge research fields, studying how to define neural networks for graph processing that do not rely on backpropagation for their training. The goal of our research is to develop an extremely efficient model for graphs. The training phase of most of the GNN architectures that have been proposed in the literature in the last years is considerably computation-demanding. One of the reasons is that they rely on the backpropagation algorithm. Defining a backpropagation-free architecture reminiscent of the idea that each neuron can be trained independently allows us to have a very simple and highly parallelizable model that significantly reduces the temporal demands of the training phase. Moreover, it is also open to the application of GNNs to more flexible scenarios where the model can be updated continuously during its use. This study aims to investigate the development of a backpropagation-free architecture for graphs and show how the use of a different training approach affects the performance of the GNN, without compromising its effectiveness.

Our exploration is based on the recently proposed Gated Linear Networks (GLN) [4], a family of backpropagation-free neural networks that have been developed for online learning and that have shown promising results. The main characteristic of such networks is that, contrary to the mainstream approach, the nonlinearity is achieved via a gating mechanism instead of element-wise nonlinear functions. More specifically, each neuron receives a context vector as additional input that is used to select one weight vector in a pre-defined set. The only nonlinearity lies in such a gating mechanism. In fact, once the weight vector is selected, each neuron behaves linearly. The resulting network is a piece-wise linear model (similar to ReLU networks). While the gating mechanism is not trained, each neuron learns to predict

(by modifying the weights) a binary output and can be trained independently from the rest of the network. In the case of multi-class classification, a one-vs-all approach is exploited.

In order to define neural networks capable of processing graph data, we explore two possible generalizations of the above mechanism. The first one is based on the approach adopted by many graph convolutional networks, in which the network architecture reflects the structure of the input graph, and node representations are refined at each layer according to the local graph topology via an aggregation operation over neighboring nodes. For this generalization, we also provide a theoretical result on the incremental expressiveness of our models. The second one is inspired by the so-called multi-resolution architecture on which the graph convolutional layer is defined by exploiting the power series of the diffusion operator (also known as graph-augmented multi-layer perceptrons (GA-MLPs) [7], or polynomial-based graph convolutional (PGC) [8]). These models are able to simultaneously and directly consider all topological receptive fields up to k-hops. Moreover, applying the gating mechanism turns out to be decoupled from the graph topology.

The properties inherited from GLNs ensure that our models are (at least in principle) as expressive as their backpropagation-based counterparts, with a significantly easier training phase. Nonetheless, several choices have to be made, such as which neighbor aggregation mechanism to adopt, how to define the contexts on graphs and how to define the gating mechanism efficiently on graphs to limit the number of parameters of the network while obtaining good predictive performances.

We experimentally evaluate our backpropagation-free graph convolutional neural architectures on commonly adopted node classification benchmarks and verify their competitive performance. This work paves the way for novel neural approaches for graph learning. This is an extended version of the work [9]. Compared to the conference version, we explored a novel gating mechanism for GLNs, namely the soft-gating approach. We extended our theoretical analysis by incorporating both the soft- and hard-gating mechanisms in a unified framework. We extended the experimental results, including the new gating mechanism, and we analyzed the impact of the backpropagation-free models for what concerns computational efficiency. Specifically, we explored the speed-up that the BF-GNNs bring to the training phase and how their particular structure helps in reducing the cost of the model selection process.

In this paper we explore how to define a GNN that can be trained faster and more efficiently than using backpropagation. Our proposed solution involves constructing an architecture where each neuron can be trained independently, resulting in a highly parallelizable model that reduces the time required for training and inference. The results and analysis show that using a backpropagation-free model is more efficient while maintaining comparable performance to backpropagation-based models.

2 Background

This section introduces the background notation and knowledge on which our model hinges.

2.1 Learning on graph nodes

A learning problem on a graph can be formulated as learning a function that maps nodes to labels. The underlying graph structure is given as $G = (V, E, \mathcal{L})$, where $V = \{v_1, \ldots, v_n\}$ is the set of nodes, $E \subseteq V \times V$ is the set of edges connecting the nodes, and $\mathcal{L} : \mathcal{V} \to \mathbb{R}^s$ is a function associating a vector of attributes to each node. With $\mathcal{N}(v)$ we denote the set of

nodes adjacent to v, i.e., $\mathcal{N}(v) = \{u \mid (v, u) \in E\}$. To simplify the notation, we define for a fixed graph G the matrix $\mathbf{X} = [\mathcal{L}(v_1), \ldots, \mathcal{L}(v_n)]^\intercal$

Given a graph G, our training set is composed of the target information associated with some of the graph nodes, i.e., $\{(v, y) \mid v \in W, y \in \mathcal{Y}\}$ with $W \subset V$. For the sake of simplicity, in our presentation, we will only consider binary values $\mathcal{Y} \in \{0, 1\}$.

2.2 Graph neural networks

Although learning from graph data is not a new research field [10–12], in the last few years, graph neural networks have emerged as the machine learning model of choice when dealing with graph problems. The first definition of neural network for structured data, including graphs, has been proposed by Sperduti and Starita [10]. Later, it has been refined by Micheli [13] and Scarselli et al. [12]. The core idea is to define a neural architecture that is modeled according to the graph topology. Thanks to weight sharing, the same set of neurons is applied to each vertex in the graph and computes its output based on the representation of the vertex and of its neighbors. As usual, the function computed by each layer is parametric. A graph neural network (GNN) is a neural model that exploits the structure of the graph and the information embedded in feature vectors of each node to learn a representation $\mathbf{h}_v \in \mathbb{R}^m$ for each vertex $v \in V$. In many GNN models, the computation of $\mathbf{h}_v$ can be divided into two main steps: *aggregate* and *combine*. We can define aggregation and combination by using two functions, $\mathcal{A}$ and $\mathcal{C}$, respectively: $\mathbf{h}_v = \mathcal{C}(\mathcal{L}(v), \mathcal{A}(\{\mathcal{L}(u) : u \in \mathcal{N}(v)\}))$.

The choice of aggregation function $\mathcal{A}$ and combination function $\mathcal{C}$ defines the type of *graph convolution (GC)* adopted by the GNN. In more detail, a general graph neural network model is built according to the following equations. First, d graph convolution layers are stacked: $\mathbf{h}_v^{(i)} = f(\text{graphconv}(\mathbf{h}_v^{(i-1)}, \{\mathbf{h}_u^{(i-1)} | u \in \mathcal{N}(v)\})$, where $f(\cdot)$ is an element-wise nonlinear activation function, $graphconv(\cdot, \cdot)$ is a graph convolution operator, $\mathbf{h}_v^{(i)}$ is the representation of node v at the i-th graph convolution layer, $1 \leq i \leq d$, and $\mathbf{h}_v^{(0)} = \mathbf{x}_v$ (i.e., the row of $\mathbf{X}$ corresponding to v). The mechanism of communication between neighboring nodes exploited by the convolution operator is dubbed *message passing*.

In the last few years, several different GCs have been proposed [14–20].

This work builds on top of two widely adopted graph convolutions. The first one is the GCN [14]

$$\mathbf{H}^{(i)} = \mathcal{F}\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I} + \mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}}\,\mathbf{H}^{(i-1)}\mathbf{W}^{(i)}\right), \quad i > 1 \tag{1}$$

where $\mathbf{A}$ denotes the standard adjacency matrix of the graph G and $\tilde{\mathbf{D}}$ a diagonal degree matrix with the diagonal elements defined as $\tilde{d}_{ii} = 1 + \sum_j a_{ij}$. Further, $\mathbf{H}^{(i)} \in \mathbb{R}^{n \times m_i}$ is a matrix containing the representation $\mathbf{h}_v^{(i)}$ of all nodes in the graph (one per row) at layer i, $\mathbf{W}^{(i)} \in \mathbb{R}^{m_{i-1} \times m_i}$ denotes the matrix of the layer's parameters, and $\mathcal{F}$ is the element-wise (usually, nonlinear) activation function.

The second graph convolution we consider is a slight variation of the first model and commonly referred to as GraphConv [2]:

$$\mathbf{h}_v^{(i)} = \mathcal{F}\left(\mathbf{h}_v^{(i-1)}\mathbf{W}_1^{(i)} + \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(i-1)}\mathbf{W}_2^{(i)}\right),$$

where $\mathbf{W}_1^{(i)}, \mathbf{W}_2^{(i)} \in \mathbb{R}^{m_{i-1} \times m_i}$ (with $m_0 = s$, the input dimensionality) are the network parameters.

2.3 Multi-resolution GNN

Some recent works in the literature exploit the idea of extending graph convolution layers to increase the receptive field size without increasing the depth of the model. The basic idea underpinning these methods is to consider the case in which the graph convolution can be expressed as a polynomial of the powers of a transformation $\mathcal{T}(\cdot)$ of the adjacency matrix. The models based on this idea can simultaneously and directly consider all topological receptive fields up to l-hops, just like the ones that are obtained by a stack of graph convolutional layers of depth l, without incurring in the typical limitations related to the complex interactions among the parameters of the GC layers. Formally, the idea is to define a representation as built from the contribution of all topological receptive fields up to l-hops as:

$$\mathbf{H} = f(\mathcal{T}(\mathbf{A})^0\mathbf{X}, \mathcal{T}(\mathbf{A})^1\mathbf{X}, \ldots, \mathcal{T}(\mathbf{A})^l\mathbf{X}), \tag{2}$$

where $\mathcal{T}(\cdot)$ is a transformation of the adjacency matrix, (e.g., the Laplacian matrix), and f is a function that aggregates and transforms the various components obtained from the powers of $\mathcal{T}(\mathbf{A})$, for instance, the concatenation, the summation, or even something more complex, such as a multi-layer perceptron. The f function can be defined as a parametric function, depending on a set of parameters $\boldsymbol{\theta}$ whose values can be estimated from data (e.g., when f involves an MLP). Various models that rely on this idea have been proposed in the last few years [7, 8, 21–25].

2.4 Backpropagation-free neural networks

In this paper, we exploit recently defined neurons that can be trained locally and independently instead of using backpropagation. We consider the recently proposed Gated Linear Networks [26] (GLNs) where the local optimization problem obtained for each neuron, adopting an appropriate loss, is convex. Moreover, it has been shown that GLNs can represent any function that represents a probability arbitrarily well [27]. The aim of GLNs is to define a model composed of neurons that can be trained locally, independently and relying only on task supervision. The main difference between GLNs and MLPs trained with backpropagation is that, in GLNs, the weight update of neurons in a layer does not depend on the following ones. Each neuron is trained to predict the target value and can be trained independently from the rest of the network (provided the input). GLNs have been applied to online and continual learning problems [28].

Each neuron in a GLN is a Gated Geometric Mixer. Geometric mixing [29] is an ensemble technique that assigns a weight to each weak predictor in input. In GLNs, every unit produces in output its prediction for the target. Given an input vector of probabilities $\mathbf{p} = [p_1, \ldots, p_n]^\top$, geometric mixing is defined as:

$$\sigma\left(\mathbf{w}^\top \sigma^{-1}(\mathbf{p})\right) \tag{3}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, $\sigma^{-1}(x) = \text{logit}(x) = \log(x) - \log(1-x)$ is the logit function (that is the inverse of the sigmoid function), and both of them are applied element-wise.

To achieve nonlinearity, specifically piecewise-linearity, GLNs employ a gating mechanism in each neuron. Each neuron divides its input space in *regions*. A geometric mixing (that is, a linear model) is associated with each region. The association from examples to regions is carried by a *region assignment* function c. GLNs assume that for each example, we have a

vectorial representation available, $\mathbf{x} \in \mathbb{R}^{d^{(x)}}$, and a vector representing side information (or context), i.e., $\mathbf{z} \in \mathbb{R}^{d^{(z)}}$. The c function is defined depending on side information associated with each input (in case no side information is available, it is possible to set $\mathbf{z} = \mathbf{x}$). Each neuron in a GLN solves a convex problem and is trained independently to predict the target.

For the sake of simplicity, we omit bias terms in the following formulations. Given a neuron j at the i-th layer, its output is defined as:

$$\mathbf{h}^{(i)}_{j,(\mathbf{x},\mathbf{z})} = \sigma\left(\sigma^{-1}\left(\mathbf{h}^{(i-1)}_{(\mathbf{x},\mathbf{z})}\right)^{\top} \mathbf{w}^{(i)}_{j,(\mathbf{z})}\right), \quad i > 1 \tag{4}$$

with $\mathbf{h}^{(0)}_{(\mathbf{x},\mathbf{z})} = \sigma(\mathbf{x})$. The vector $\mathbf{w}^{(i)}_{j,(\mathbf{z})} \in \mathbb{R}^{m_{i-1}}$ stores the weights associated to the region activated by the context $\mathbf{z}$ for the corresponding neuron. Let us discuss this weight vector in more detail and the gating mechanism that defines how a specific set of weights is selected.

Given an example $(\mathbf{x}, \mathbf{z})$, we can select the weights of a single neuron j at the i-th layer as:

$$\mathbf{w}^{(i)}_{j,(\mathbf{z})} = \left(\boldsymbol{\Theta}^{(i)}_{j} \mathbf{c}^{(i)}_{j,(\mathbf{z})}\right) \tag{5}$$

where $\boldsymbol{\Theta}^{(i)}_{j} \in \mathbb{R}^{m_{i-1} \times k}$, k is the number of regions (we assume for simplicity that each neuron in the network considers the same number of regions), and $\mathbf{c}^{(i)}_{j,(\mathbf{z})} \in \mathbb{R}^{k}$. Notice that the main characteristic of a Gated Linear Neuron is that, instead of having a single weight vector, each GL neuron depends on a *matrix* of parameters $\boldsymbol{\Theta}^{(i)}_{j}$.

The original paper [26] proposes to implement the gating in the c functions with a halfspace-gating mechanism. Given a vector $\mathbf{z} \in \mathbb{R}^{d^{(z)}}$, and a hyperplane with parameters $\mathbf{a}_i \in \mathbb{R}^{d^{(z)}}$ and $b_i \in \mathbb{R}$, let us define a context function $\tilde{c}_i : \mathbb{R}^{d^{(z)}} \to \{0, 1\}$ as:

$$\tilde{c}_i(\mathbf{z}) = \begin{cases} 1 & \text{if } \mathbf{a}_i^{\top}\mathbf{z} > b_i \\ 0 & \text{otherwise} \end{cases}$$

that divides $\mathbb{R}^{d^{(z)}}$ in two half-spaces, according to the hyperplane $\mathbf{a}_i^{\top}\mathbf{z} = b_i$. We can compose $\log_2(k)$ (assuming k to be a power of 2) context functions of the same kind, obtaining a higher-order context function $\tilde{c} : \mathbb{R}^{d^{(z)}} \to \{0, 1\}^{\log_2(k)}$, $\tilde{c} = [\tilde{c}_1, \ldots, \tilde{c}_k]^{\top}$. We can then easily define a function f mapping from $\{0, 1\}^{\log_2(k)}$ to $\{0, \ldots, k-1\} \subset \mathbb{N}$, obtaining the function $\hat{c} : \mathbb{R}^{d^{(z)}} \to \{0, \ldots, k-1\}$, $\hat{c} = f \circ \tilde{c} = f(\tilde{c}(\mathbf{z}))$. We can exploit the one-hot encoding of the output of such function and re-define it as $c : \mathbb{R}^{d^{(z)}} \to \{0, 1\}^{k}$, $c = \text{one\_hot}(\hat{c})$.

Given a layer i, each neuron j computes a different function $c^{(i)}_{j} : \mathbb{R}^{d^{(z)}} \to \{0, 1\}^{k}$. For the j-th neuron at the i-th layer, the output of the context function applied to $\mathbf{z}$ is thus the (one-hot) vector $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$. Notice that this is a *hard-gating* mechanism, in the sense that fixed a context vector, the gating function will select a single weight from the set of weights associated with the Gated Linear Neuron.

2.4.1 Layer-wise formulation

We can define a whole GLN layer by exploiting the definition of a single Gated Linear Neuron in the previous section. This formulation will be exploited in the remainder of the paper. The output for the i-th layer in a GLN (with m_i neurons) for a sample $(\mathbf{x}, \mathbf{z})$ is defined as:

$$\mathbf{h}^{(i)}_{(\mathbf{x},\mathbf{z})} = \sigma\left(\sigma^{-1}\left(\mathbf{h}^{(i-1)}_{(\mathbf{x},\mathbf{z})}\right)^{\top} \mathbf{W}^{(i)}_{(\mathbf{z})}\right), \quad i > 1 \tag{6}$$

where

$$\mathbf{W}_{(\mathbf{z})}^{(i)} = \left[\mathbf{w}_{1,(\mathbf{z})}^{(i)}, \ldots, \mathbf{w}_{m_i,(\mathbf{z})}^{(i)}\right], \tag{7}$$

and $\mathbf{W}_{(\mathbf{z})}^{(i)} \in \mathbb{R}^{m_{i-1} \times m_i}$.

Several layers can then be stacked. For a binary classification problem, the last layer will comprise a single neuron, i.e., for a network with l layers, we have: $\mathbf{W}_{(\mathbf{z})}^{(l)} = \mathbf{w}_{1,(\mathbf{z})}^{(l)}$, $\mathbf{W}_{(\mathbf{z})}^{(l)} \in \mathbb{R}^{m_{l-1} \times 1}$. The resulting model is, by construction, piecewise-linear. Specifically, given a context $\mathbf{z}$, the model is (up to a final activation function) linear and can be written as:

$$y_{(\mathbf{x},\mathbf{z})} = \sigma\left(\mathbf{x}^\top \mathbf{W}_{(\mathbf{z})}^{(1)} \ldots \mathbf{W}_{(\mathbf{z})}^{(l-1)} \mathbf{W}_{(\mathbf{z})}^{(l)}\right) = \sigma\left(\mathbf{x}^\top \mathbf{w}_{(\mathbf{z})}\right) \tag{8}$$

with a weight vector $\mathbf{w}_{(\mathbf{z})} \in \mathbb{R}^{d^{(x)}}$.

3 Backpropagation-free graph neural networks

This section defines our proposed models, which generalize GLNs to graph-structured data. Firstly, we show how to embed the GLN idea into graph convolutions to build models based on the message-passing paradigm. Then, we propose a multi-resolution approach that keeps the features' propagation through the graph structure separate from the processing of the resulting node representations.

3.1 Message-passing GLN

The core concept behind several definitions of graph neural networks is the aggregation function used to obtain information about the local graph structure surrounding a graph node. The most straightforward aggregation mechanism involves just the summation of neighboring nodes' representations. For this simple mechanism, we obtain the following definition for a single layer in a Gated Linear Graph Neural Network:

$$\mathbf{h}_{(v,\mathbf{z})}^{(i)} = \sigma\left(\sigma^{-1}\left(\mathbf{h}_{(v,\mathbf{z})}^{(i-1)}\right)^\top \mathbf{W}_{(\mathbf{z})}^{(i,1)} + \left(\sum_{(u,\mathbf{z}') \in \mathcal{N}_v} \sigma^{-1}\left(\mathbf{h}_{(u,\mathbf{z}')}^{(i-1)}\right)^\top \mathbf{W}_{(\mathbf{z}')}^{(i,2)}\right)\right), \tag{9}$$

for $i \geq 1$, and $\mathbf{h}_{v,\mathbf{z}}^{(0)} = \sigma(\mathcal{L}(v))$. The weights $\mathbf{W}_{(\mathbf{z}')}^{(i,2)}$ and $\mathbf{W}_{(\mathbf{z})}^{(i,1)}$ are defined as per Eq. (7) and can be obtained by backpropagation-free training. This model can be considered as a modification of GraphConv proposed in [2] in which gated geometric mixing has been applied. For this reason, we refer to this model as Backpropagation-Free-GraphConv (BF-GraphConv). Similarly to common formulations of graph neural networks, we can express the hidden representation for all the nodes in the graph as a single matrix. We obtain the following form of BF-GraphConv:

$$\mathbf{H}_{(\mathbf{z})}^{(i)} = \sigma^{-1}\left(\mathbf{H}_{(\mathbf{z})}^{(i-1)}\right)\mathbf{W}_{(\mathbf{z})}^{(i,1)} + \mathbf{A}\sigma^{-1}\left(\mathbf{H}_{(\mathbf{z})}^{(i-1)}\right)\mathbf{W}_{(\mathbf{z})}^{(i,2)} \tag{10}$$

where $\mathbf{H}_{(\mathbf{z})}^{(i)} \in \mathbb{R}^{n \times m_i}$ and $\mathbf{H}_{(\mathbf{z})}^{(0)} = \sigma(\mathbf{X})$. BF-GraphConv can therefore be regarded as a piecewise linear GNN depending on the neurons' context information $\mathbf{z}$.

Following common definitions of graph neural networks, we can resort to any message-passing mechanism and define the Gated Linear counterpart. For instance, we can also

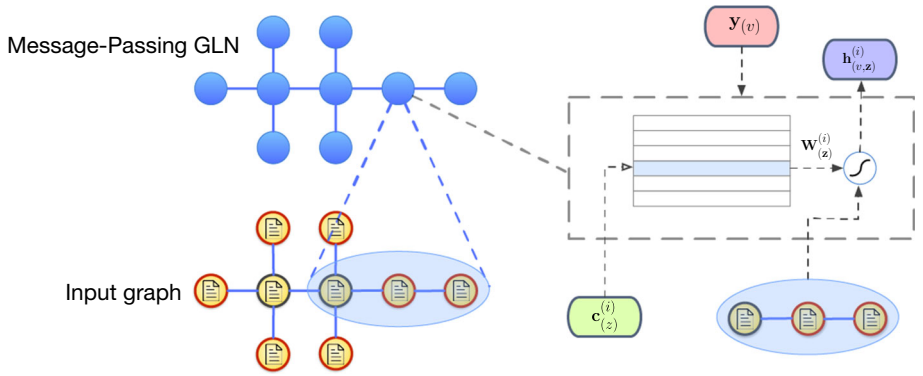


Fig. 1 A graphical layout of the proposed Message-Passing GLN, with an expanded view of the GLN neuron. Each neuron receives in input the embedding of a node and the corresponding context $\mathbf{c}^{(i)}_{(\mathbf{z})}$ (used to select the associated weights $\mathbf{W}^{(i)}_{(\mathbf{z})}$) and produces in output the value $\mathbf{h}^{(i)}_{(\mathbf{v},\mathbf{z})}$, while $\mathbf{y}_{(v)}$ represents the target supervision

consider the GCN presented in Eq. (1) which leads to the following BF-GCN:

$$\mathbf{H}^{(i)}_{(\mathbf{z})} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I}+\mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}}\,\sigma^{-1}\left(\mathbf{H}^{(i-1)}_{(\mathbf{z})}\right)\mathbf{W}^{(i)}_{(\mathbf{z})}\right). \tag{11}$$

The resulting model can be regarded as a piecewise linear GCN. In particular, after l layers, the output $\mathbf{H}^{(l)}_{(\mathbf{z})}$ for the context $\mathbf{z}$ can be written as

$$\mathbf{H}^{(l)}_{(\mathbf{z})} = \sigma\left(\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I}+\mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}}\right)^{l}\mathbf{X}\mathbf{w}_{(\mathbf{z})}\right), \tag{12}$$

i.e., the BF-GCN model with l layers is a generalization of the simple graph convolutional network (SGC) introduced in [21] and further investigated in [30], where the vector of weights $\mathbf{w}_{(\mathbf{z})}$ changes based on the input context. Notice that the main differences between the Gated Linear Graph Neural Networks and commonly adopted GNN formulations are the local training and the gating mechanism. A graphical layout of the proposed architecture is reported in Fig. 1. The figure also reports an expanded view of the GLN neuron exploited to define the BF-GCN (and the other architectures proposed in this paper).

3.2 Multi-resolution GLN

A different definition of GNN is based on the idea of exploiting the power series of the diffusion operator to obtain a multi-scale representation of the graph features. The obtained representation is usually fed to an MLP that projects it into the output space. Our proposal is to substitute the MLP with a GLNs architecture.

Considering the explorative purpose of this work, we decide to adopt the most general multi-scale representation definition proposed in [8]:

$$\mathbf{R}_{l,\mathcal{T}} = [\mathbf{X}, \mathcal{T}(\mathbf{A})\mathbf{X}, \mathcal{T}(\mathbf{A})^2\mathbf{X}, \ldots, \mathcal{T}(\mathbf{A})^l\mathbf{X}],$$

where $\mathcal{T} : \bigcup_{j=1}^{\infty}(\mathbb{R}^{j\times j} \to \mathbb{R}^{j\times j})$ is a generic transformation of the adjacency matrix that preserves its shape, i.e., $\mathcal{T}(\mathbf{A}) \in \mathbb{R}^{n\times n}$. For instance, $\mathcal{T}$ can be defined as the function returning the Laplacian matrix starting from the adjacency matrix. Then we can apply the

GLN-based classifier. In particular, for each degree of diffusion operator (up to l), we consider c neurons, where c is the number of the classes considered in the classification problem. We recall that each neuron solves a binary classification problem. Therefore, similar to the message-passing GLN case, the one-vs-rest approach is exploited.

$$\begin{aligned}\mathbf{h}_i &= [\mathbf{h}^{(i)}_{j,(\mathbf{R}_{i,\mathcal{T}},\mathbf{X})}]_{\forall j\in[0\ldots c-1]}\\ \mathbf{h} &= [\text{GeomMean}(\mathbf{h}_0[j],\ldots,\mathbf{h}_l[j])]_{\forall j\in[0\ldots c-1]}.\end{aligned} \tag{13}$$

Note that $\mathbf{h}_i$ is computed considering only the diffusion operator's multi-resolution representation up to degree i. This allows us to obtain the same effect that the authors have in [8] where l multi-resolution convolutions of degree ranging from 0 up to l are concatenated. Finally, to combine the results of each class's neurons, we compute the geometric mean.

3.3 Context functions

The context function presented in Sect. 3 and exploited in (5) is based on random half-space gating. That definition is suited for online learning, where the training data distribution is not known beforehand. However, it is not data-driven and may result in the necessity of defining a high number of context regions to obtain a sufficiently nonlinear model. Notice that the halfspace gating mechanism depends on some hyperparameters: in addition to the number of regions k, one has to choose the parameters of the distribution from which to sample the weights corresponding to each hyperplane (e.g., mean and variance, assuming they are sampled from a normal distribution). Setting these hyperparameters may be challenging since their choice can strongly affect results.

In this section, we propose an alternative approach that can be exploited in the batch learning scenario and that does not depend on any parameter but on the number of regions to consider.

In particular, we propose to define a partition of the context space based on a set of prototypes [28]. Each point in the space is assigned to its closest prototype, obtaining a Voronoi tessellation. Note that half-space gating generates a division of the context space that can be represented as a planar straight-line graph (PSLG) instead. It is possible to show that any PSLG coincides with the Voronoi diagram of some set S of points (i.e., prototypes) [31]. Similarly to the half-space gating mechanism, the prototypes are not learned. However, instead of randomly generating them, we propose sampling them randomly among the training examples. This ensures that each prototype will lie on the input data manifold. Moreover, as mentioned before, this approach relieves us from many hyperparameter choices.

Let $\mathbf{P}^{(i)}_j \in \mathbb{R}^{k\times d^{(\mathbf{z})}}$ be the matrix of prototypes. We can formally define for every $z \in \mathbb{R}^{d^{(z)}}$ the context vector $\mathbf{c}^{(i)}_{j,(\mathbf{z})} \in \{0,1\}^k$ as:

$$\mathbf{c}^{(i)}_{j,(\mathbf{z})} = \text{one\_hot}(\text{argmin}(\text{vecnorm}(\mathbf{P}^{(i)}_j - \mathbf{1}\otimes\mathbf{z}^\top))),$$

where $\mathbf{1}$ is a vector with all k elements equal to 1, vecnorm($\mathbf{M}$) returns the 2 norm of each row of $\mathbf{M}$, and $\otimes$ is the Kronecker product.

3.4 A unified framework for soft and hard gating

A one-hot function $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ assigns to every context vector $\mathbf{z}$ precisely one region in $\mathbb{R}^{d^{(z)}}$ based on the closest distance to one of the prototypes. The prototypes correspond to the k centers

of the Voronoi regions. For a one-hot vector $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ exactly one of the entries is one (the one corresponding to the closest prototype), while all other entries are zero. This implies that only the closest prototype is considered when computing the output of the neurons, regardless of the distances of $\mathbf{z}$ to the other prototypes. In the following, we refer to this selection of $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ as hard gating. Hard gating can lead to discontinuities in the outputs when the context $\mathbf{z}$ is similarly close to several prototypes and the weights of the different prototypes vary. In this case, a small amount of noise in the context $\mathbf{z}$ can lead to considerably different neuron outputs. To diminish these discontinuities on the borders of the regions, we alternatively propose a soft-gating procedure.

Soft gating uses a more general probability distribution for the vectors $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ instead of a one-hot vector. In fact, one-hot vectors can be interpreted as a point mass distribution. We will construct these general probability distributions in such a way that the entries of the vectors $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ depend smoothly on the distance to the prototypes and get large if the respective distance is small. Using a probability distribution for the vectors $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ results, according to Eq. (5), to a weight $\mathbf{w}^{(i)}_{j,(\mathbf{z})}$ which is a weighted sum of the contributions of all the weights associated with the single prototypes.

For a soft-gating procedure, we, therefore, use probabilistic assignment vectors with the following properties for every neuron j and every layer i:

$$\mathbf{c}^{(i)}_{j,(\mathbf{z})} \in [0,1]^k \quad \text{and} \quad \|\mathbf{c}^{(i)}_{j,(\mathbf{z})}\|_1 = 1 \quad \text{for all } \mathbf{z} \in \mathbb{R}^{d^{(z)}}.$$

Remark 1 (i) The one-hot functions considered in the previous part are special cases of this definition. One-hot functions allow linking the assignment functions to regions with the prototypes as centers.

(ii) The set of all entries of the vectors $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ considered as functions in $\mathbf{z}$ forms a so-called partition of unity for the Euclidean space $\mathbb{R}^{d^{(z)}}$. This allows to distribute the contributions of the single prototypes to the combined weight once the context vector $\mathbf{z}$ is given. In hard gating, this partition of unity corresponds to a segmentation of $\mathbb{R}^{d^{(z)}}$ in k disjoint regions.

(iii) From a different point of view, we can consider the entries of $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ as function values that depend on the neuron j. In this particular point of view, the entries of $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ form a partition of unity on the network. In [32], it was shown that the partition of unities based on overlapping covers of the network has advantages in the reconstruction of smooth functions compared to the partition of unities that are derived from pure segmentation of the network.

Soft-gating assignment vectors $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ can be generally constructed using auxiliary vectors $\psi^{(i)}_{j,\mathbf{z}}$ with positive entries that depend continuously on the Euclidean distance to the prototypes. Then a probabilistic assignment vector can be easily obtained by normalizing the auxiliary vectors, i.e., by setting:

$$\mathbf{c}^{(i)}_{j,(\mathbf{z})} = \frac{\psi^{(i)}_{j,\mathbf{z}}}{\|\psi^{(i)}_{j,\mathbf{z}}\|_1}.$$

Being defined as a probabilistic vector is not enough for $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ to encode a meaningful measure between an example and the prototypes. Specifically, we are interested in definitions encoding the *similarity* between examples and prototypes. There are multiple possibilities to define such a measure. In the following, we report the one we have used for our experiments.

Aiming to define a similarity and not a distance measure, we can simply subtract the normalized distance from 1:

$$\mathbf{c}^{(i)}_{j,(\mathbf{z})} = \mathbf{1} - \frac{\psi^{(i)}_{j,\mathbf{z}}}{\|\psi^{(i)}_{j,\mathbf{z}}\|_1},$$

where $\psi^{(i)}_{j,\mathbf{z}}$ can be instantiated as a function computing the Euclidean distance between the context vector $\mathbf{z}$ and the matrix $\mathbf{P}^{(i)}_j$ of the prototypes of the j-th neuron of layer i, i.e.,

$$\psi^{(i)}_{j,\mathbf{z}} = \text{vecnorm}(\mathbf{P}^{(i)}_j - \mathbf{1} \otimes \mathbf{z}^\top).$$

Using this formulation, and according to Eq. (5), $\mathbf{w}^{(i)}_{j,(\mathbf{z})}$ becomes the weighted sum of the contributions of all input projections obtained considering the weights associated with each region. The weight of each projection is based on the distance (the closer—the higher). Note also that this formulation ensures that each element in $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ is comprised in the interval [0, 1], and $|\mathbf{c}^{(i)}_{j,(\mathbf{z})}| = 1$. Since this is the simplest among the options to implement a soft $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$, we consider this formulation in our experiments.

3.5 Incremental expressivity of soft Gated Linear Networks

The context functions $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$ incorporated in soft gating depend continuously on the Euclidean distance from the contexts $\mathbf{z} \in \mathbb{R}^{d^{(z)}}$ to the prototype vectors encoded in the matrix $\mathbf{P}^{(i)}_j$. Now, if the number of prototypes gets larger, we expect the resulting GLNs to become more expressive. We formalize and show this expected behavior for the two gated linear networks, BF-GCN and BF-GraphConv. It is true for the soft and hard gated variants of the networks.

Theorem 1 *We consider two Gated Linear Graph Neural Networks (either BF-GCN or BF-GraphConv) based on a soft-gating assignment function $\mathbf{c}_{(\mathbf{z})}$. We assume that the second network contains more prototypes than the first network, i.e., we assume that the set $\mathcal{P}_2 \subset \mathbb{R}^{d^{(z)}}$ of prototypes of the second network model contains the prototypes $\mathcal{P}_1$ of the first model. Then, the second Gated Linear GNN is more expressive than the first.*

Proof In this proof, we will only consider the BF-GCN model. For simplicity, we will also assume that $\mathbf{X} = \mathbf{x} \in \mathbb{R}^{n\times 1}$, i.e., that the dimension of the input variable is $s = 1$ and, thus, that we have real-valued weights $w_{(\mathbf{z})} \in \mathbb{R}$. Now, for a set $\mathcal{P}$ of prototypes let $\mathcal{H}(\mathcal{P})$ denote the set of all possible functions

$$\mathcal{H}(\mathcal{P}) = \left\{ \mathbf{y}_{(\mathbf{z})} = \sigma\left((\tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I}+\mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}})^l \, \mathbf{x}\, w_{(\mathbf{z})} \right) \mid w_{(\mathbf{z})} = \sum_{\mathbf{p}\in\mathcal{P}} \mathbf{c}^{(\mathcal{P})}_{(\mathbf{z}),\mathbf{p}} w_{(\mathbf{p})} \right\}$$

generated by a BF-GCN network with a given assignment vector $\mathbf{c}^{(\mathcal{P})}_{(\mathbf{z})}$ with entries $\mathbf{c}^{(\mathcal{P})}_{(\mathbf{z}),\mathbf{p}}$ and set $\mathcal{P}$ of prototypes. Assume now that the weight $w_{(\mathbf{z})}$ is generated as a positive combination of the weights of the prototypes in the smaller set $\mathcal{P}_1$, i.e.,

$$w_{(\mathbf{z})} = \sum_{\mathbf{p}\in\mathcal{P}_1} \mathbf{c}^{(\mathcal{P}_1)}_{(\mathbf{z}),\mathbf{p}} w_{(\mathbf{p})}.$$

Table 1 Dataset statistics

Dataset	#Classes	#Edges	#Train	#Val	#Test
Citeseer	6	9228	1995	666	666
Cora	7	10,556	1624	542	542
Pubmed	3	88,651	11,829	3944	3944
WikiCS	10	216,123	7021	2340	2340

The columns #Train, #Val and #Test report the number of nodes in the training, validation and test sets, respectively

By setting $\hat{w}_{(\mathbf{p})} = w_{(\mathbf{p})}\big(\sum_{\mathbf{p}\in\mathcal{P}_1} \mathbf{c}^{(\mathcal{P}_2)}_{(\mathbf{z}),\mathbf{p}}\big)^{-1}$ if $p \in \mathcal{P}_1$ and $\hat{w}_{(\mathbf{p})} = 0$ for $p \in \mathcal{P}_2 \backslash \mathcal{P}_1$, we get

$$w_{(\mathbf{z})} = \sum_{\mathbf{p}\in\mathcal{P}_2} \mathbf{c}^{(\mathcal{P}_2)}_{(\mathbf{z}),\mathbf{p}} \hat{w}_{(\mathbf{p})},$$

and, therefore that $\mathcal{H}(\mathcal{P}_1) \subseteq \mathcal{H}(\mathcal{P}_2)$. □

Note that, in order to demonstrate the strict inclusion $\mathcal{H}(\mathcal{P}_1) \subset \mathcal{H}(\mathcal{P}_2)$ in Theorem 1, assuming that $\mathcal{P}_2$ is strictly larger than the set $\mathcal{P}_1$, an additional assumption on the entries $\mathbf{c}^{(\mathcal{P}_2)}_{(\mathbf{z}),\mathbf{p}}$ of the context functions is required. If considered as functions of $\mathbf{z}$, the entries $\mathbf{c}^{(\mathcal{P}_2)}_{(\mathbf{z}),\mathbf{p}}$, $\mathbf{p} \in \mathcal{P}_2$, have to be linearly independent. Only in this way it can be guaranteed that additional prototypes will generate new linear combinations of weights. From Theorem 1, we also get the following result as an immediate consequence.

Corollary 1 *Every BF-GCN with soft gating and more than one prototype is more expressive than a simplified graph convolutional network (SGC)* [21]

$$\mathbf{H} = \sigma\left(\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\right)^{l}\mathbf{X}\mathbf{W}\right).$$

4 Experimental results

We exploited our proposed backpropagation-free graph neural networks on four benchmark datasets of graph node classification commonly adopted in the literature: Citeseer, Cora, Pubmed and WikiCS [33]. Citeseer, Cora and Pubmed are datasets representing relationships among research papers. The task is to classify each paper into a pre-defined set of topics. Each dataset is composed of a single large graph. The nodes represent documents and are enriched by a 0/1-valued bag-of-words features vectors indicating the absence/presence of the corresponding word from a dictionary. WikiCS is a dataset of Computer Science Wikipedia articles represented as nodes in a graph. Two nodes are connected if a hyperlink connects the two corresponding web pages. The node features are derived from the text in the web page, as the average of pre-trained *GloVe* word embeddings. For each dataset, we split the nodes into training (60%), validation (20%), and test (20%) sets. Relevant statistics on the datasets are reported in Table 1. As performance measure we opted for the classification accuracy, since it is the most common choice for the benchmark datasets we considered.

We used PyTorch Geometric [34] to develop all the models in our experimental comparison. We considered the GCN and the GraphConv convolutions as baselines, for which we exploited the implementation provided by the library. In addition, we also evaluated a multi-resolution architecture (MRGNN) as a baseline that exploits the same graph-augmented

representation of the proposed backpropagation-free multi-resolution GLN, followed by an MLP that projects the node representations to the output space. All the baseline models are trained end-to-end using the backpropagation algorithm.

We explored two choices for the definition of context of the multilayer backpropagation-free models (BF-GCN and BF-GraphConv), i.e., the standard definition in which the context is the same as the input $\mathbf{X}$ and an alternative formulation where we use each node's inputs, i.e., the hidden representation computed at the preceding layer ($\mathbf{H}$) as a context for all gated neurons. Moreover, we tested our proposal of a soft-gating mechanism presented in Sect. 3.4. We used the Adam algorithm to solve the resulting optimization problems. We used early stopping (with the patience set to 15 epochs) and model checkpoint, monitoring the accuracy on the validation set. We set the maximum number of epochs to 250. All the baseline experiments involved the softmax activation function applied to the last layer. The results were averaged by performing five runs for each model. Our experiments have been executed on a machine equipped with: 2x Intel(R) Xeon(R) CPU E5-2630L v3, 192 GB of RAM and an Nvidia Tesla V100. For more details, please check the publicly available code.[1]

4.1 Model selection

Before discussing the results of our experiment, it is of utmost importance to stress that different papers in the literature adopt different model selection and error estimation procedures. Different procedures can produce very different results. A key aspect to consider is the procedure adopted to select the hyperparameters (such as learning rate, regularization, network architecture). Many papers report, for each dataset, the best performance (on the test set) obtained after testing many hyperparameter configurations. This procedure favors complex methods that depend on many hyperparameters, since they have a larger set of trials to select from compared to simpler methods. However, the predictive performances computed in this way are not unbiased estimations of the true error; thus, these results are not comparable to other model selection methods [35]. Moreover, since neural networks tend to show long training times, different works propose to fix different hyperparameters depending on the particular datasets considered. To avoid differences in results due to the model selection conditions, we ran all the experiments using the same fair model selection procedure, where we selected all the hyperparameters of each method on the validation set. The hyperparameters of the model (number of hidden units, number of layers, learning rate, weight decay, and dropout) were selected by using a narrow grid search, where the explored sets of values do change based on the considered dataset. We performed preliminary tests to select the set of values considered for each hyperparameter. In Table 2, we report the sets of hyperparameter values used for the grid search. For what concerns the selection of the epoch, it was performed for each fold independently based on the accuracy value on the validation set. The importance of the validation strategy is discussed in [36], where results of a fair comparison among the considered baseline models are reported. We use the same hyperparameter grid for all the models to perform a fair comparison between the proposed models and the baselines. Since our models adopt a one-vs-rest approach, for the baselines we also consider a number of hidden units corresponding to the values reported in Table 2 multiplied by the number of classes of the considered dataset. As an evaluation measure to perform model selection, we used the average accuracy computed on the validation set, while we report in Tables 3, 4 and 5 the average accuracy on the test set along with the hyperparameters selected by the validation process. We recall that the contribution of this paper is not to present yet another

[1] https://github.com/lpasa/BF-GNN.

Table 2 Sets of hyperparameter values used for model selection via grid search

Hyperparameter	Values
m	2, 4, 6, 8, 16, 24, 32, 64
l	1, 2, 3, 4, 5, 6, 7, 8
k	1, 2, 4, 8, 16
Learning rate	0.1, 0.2, 0.01, 0.001
Weight decay	$0, 5^{-4}, 5^{-3}$
Drop out	0, 0.2, 0.5
$\mathcal{T}(\mathbf{A})$	$\mathbf{A}, \mathbf{L}$

Table 3 Accuracy comparison between the GCN and BF-GCN models

Model	Context function	Context	Citeseer	Cora	Pubmed	WikiCS
GCN	–	–	76.6 ± 1.0	87.5 ± 0.9	$\mathbf{88.5 \pm 0.3}$	$\mathbf{81.6 \pm 0.7}$
BF-GCN	Halfspace	**H**	76.0 ± 1.4	87.7 ± 0.5	88.1 ± 0.3	79.8 ± 0.8
			(2, 16, 2)	(1, 16, 2)	(1, 8, 2)	(2, 6, 2)
		X	76.3 ± 1.8	$\mathbf{88.0 \pm 0.5}$	88.1 ± 0.4	80.7 ± 0.7
			(2, 24, 2)	(2, 16, 2)	(2, 4, 2)	(2, 6, 8)
	Proto	**H**	$\mathbf{77.0 \pm 0.4}$	87.8 ± 0.3	88.1 ± 0.7	81.0 ± 0.9
			(2, 16, 4)	(2, 8, 2)	(2, 32, 4)	(2, 4, 16)
		X	76.9 ± 2.0	$\mathbf{88.0 \pm 0.4}$	88.0 ± 0.6	80.4 ± 0.8
			(2, 8, 1)	(2, 32, 2)	(2, 8, 2)	(2, 6, 16)
	Soft Proto	**H**	76.9 ± 1.9	87.6 ± 0.9	88.1 ± 0.4	75.8 ± 0.5
			(2, 2, 8)	(2, 24, 8)	(2, 2, 8)	(2, 6, 2)
		X	76.6 ± 2.0	87.5 ± 0.9	88.1 ± 0.4	75.9 ± 0.5
			(1, 24, 2)	(1, 24, 4)	(1, 2, 16)	(2, 6, 2)

The results in bold represent the best accuracy achieved for each dataset
The model selection is performed considering the results obtained on the validation set. Under each accuracy result, we report the hyperparameters selected via the validation process: (l, m, k)

graph neural network architecture performing slightly better than other alternatives. Instead, we want to show that it is possible to match the performance of different graph convolutional neural networks (and thus perform an effective representation learning) even without relying on backpropagation.

4.2 Discussion

The results obtained by the backpropagation-free graph neural networks, namely BF-GCN, BF-GraphConv and BF-MRGNN, are generally comparable and sometimes higher than the ones of the baselines trained with backpropagation. We recall that the main goal of backpropagation-free neural networks is not to obtain higher accuracies but to obtain comparable performance in a more parallelizable and biologically plausible setting. In particular, our proposals are significantly easier to train since they optimize convex optimization problems (one for each neuron that can be parallelized), compared to the single but significantly more complex nonlinear problem optimized by the baselines. In the following discussion

Table 4 Accuracy comparison between the GraphConv and BF-GraphConv models

Model	Context function	Context	Citeseer	Cora	Pubmed	WikiCS
GraphConv	–	–	75.1 ± 1.6	87.1 ± 0.5	$\mathbf{88.9 \pm 0.4}$	76.8 ± 1.3
BF-GraphConv	Halfspace	**H**	76.5 ± 0.9	88.0 ± 1.0	86.8 ± 0.9	80.6 ± 0.4
			(1, 32, 2)	(2, 24, 2)	(2, 2, 2)	(4, 8, 2)
		X	76.3 ± 1.8	88.0 ± 1.1	86.5 ± 0.4	80.5 ± 0.5
			(2, 24, 2)	(1, 16, 2)	(2, 8, 4)	(2, 6, 2)
	Proto	**H**	76.2 ± 1.2	$\mathbf{88.5 \pm 1.3}$	86.1 ± 0.2	81.4 ± 0.8
			(2, 8, 2)	(2, 8, 1)	(2, 32, 4)	(2, 32, 2)
		X	74.9 ± 0.6	87.8 ± 1.1	86.5 ± 0.3	$\mathbf{82.3 \pm 0.7}$
			(2, 16, 2)	(2, 8, 4)	(2, 16, 4)	(2, 8, 16)
	Soft Proto	**H**	$\mathbf{76.9 \pm 1.4}$	88.4 ± 1.2	86.5 ± 0.7	80.3 ± 0.4
			(1, 2, 32)	(2, 4, 4)	(1, 32, 16)	(1, 16, 2)
		X	76.8 ± 14.7	88.1 ± 1.4	86.1 ± 0.7	80.1 ± 0.2
			(1, 2, 64)	(1, 8, 8)	(2, 32, 2)	(2, 6, 2)

The results in bold represent the best accuracy achieved for each dataset
The model selection is performed considering the results obtained on the validation set. Under each accuracy result, we report the hyperparameters selected via the validation process (l, m, k)

Table 5 Accuracy comparison between the Backpropagation-MRGNN and standard MRGNN

Model	Context function	Citeseer	Cora	Pubmed	WikiCS
MRGNN	–	74.2 ± 1.6	86.5 ± 0.8	87.2 ± 0.9	$\mathbf{80.6 \pm 0.7}$
BF-MRGNN	Halfspace	76.3 ± 1.5	87.6 ± 0.6	87.6 ± 0.5	78.4 ± 0.7
		(3, 2)	(3, 2)	(2, 2)	(3, 4)
	Proto	$\mathbf{77.5 \pm 1.0}$	87.1 ± 0.9	$\mathbf{87.9 \pm 0.4}$	79.4 ± 0.6
		(2, 2)	(3, 2)	(2, 2)	(2, 2)
	Soft Proto	76.6 ± 1.6	$\mathbf{87.7 \pm 0.6}$	87.6 ± 0.3	77.4 ± 0.6
		(5, 4)	(3, 8)	(2, 8)	(3, 2)

The results in bold represent the best accuracy achieved for each dataset.
The model selection is performed considering the results obtained on the validation set. Under each accuracy result, we report the hyperparameters selected via the validation process (l, k)

we consider the results obtained on the test set by the model selected by validating all the hyperparameters on the validation set.

Let us start discussing the GCN convolution mechanism presented in Table 3 where we report the results obtained validating all the hyperparameters on the validation set. For each method and dataset, we report the average accuracy and the standard deviation over five runs. In the table, we compare the results of backpropagation-based GCN and the proposed backpropagation-free models (BF-GCN) with the two alternative context definitions, i.e., $\mathbf{H}$ and $\mathbf{X}$. Moreover, for BF-GCN, we experimented with the two context functions based on random half-space gating and our proposal of defining the partition of the space based on a set of prototypes (*context* column in Table 3). Finally, we report the results of the *soft-gating* mechanism (*Soft Proto*), with both choices of context function. We can notice that the performance of the backpropagation-free models is almost always close to the ones of GCN. We can conclude that, in this case, all the backpropagation-free models based on GCN can learn a representation that is comparably expressive to the one of the backpropagation-based GCN. With this convolution mechanism, there is no clear advantage of using either $\mathbf{H}$ or $\mathbf{X}$ as contexts. These results show that the proposed backpropagation-free methods are pretty resilient and show consistent performance even with significantly different choices of context space. Moreover, the results suggest that the prototype-based context function allows reaching slightly better performance in terms of accuracy compared to half-space gating.

Let us now consider the GraphConv convolution mechanism. The results of our BF methods based on this convolution are reported in Table 4. In this case, backpropagation-free models show slight but consistent accuracy improvements on the Citeseer (up to 1.8%), Cora (up to 1.4%) datasets and WikiCS (up to 5.5%). On Pubmed, BF methods perform slightly lower than the baseline GraphConv model, with a gap of at least 2.1%. Notice that the differences between BF-GraphConv and GraphConv on WikiCS and Pubmed are greater than one standard deviation. We can notice that with GraphConv, using $\mathbf{H}$ as context tends to provide slightly higher performances compared to using $\mathbf{X}$. Analyzing the accuracy, no clear advantages can be noticed in using the prototypes-based context function instead of a half-space gating mechanism.

For what concerns BF-MRGNN, the results obtained on Citeseer, Cora and Pubmed show an improvement with respect to its counterpart trained via backpropagation, i.e., MRGNN. More specifically on Citeseer, BF-MRGNN with prototypes as contexts achieved an improvement of 3.3%t compared to MRGNN, which is significant, being the two results more than

Table 6 Accuracy comparison on node classification task between the proposed models and seven baselines

Model\dataset	Citeseer	Cora	Pubmed	WikiCS
GCN	76.1 ± 1.0	87.5 ± 0.9	88.5 ± 0.3	81.6 ± 0.7
GraphConv	75.1 ± 1.6	87.1 ± 0.5	88.9 ± 0.4	76.8 ± 1.3
GAT	76.0 ± 1.4	87.3 ± 1.1	87.2 ± 0.4	79.1 ± 0.3
LGC	75.4 ± 1.3	$\mathbf{88.5 \pm 1.2}$	89.4 ± 0.8	80.0 ± 0.5
hExpGC	75.7 ± 0.80	87.5 ± 1.3	88.4 ± 0.3	78.4 ± 0.35
hLGC	77.3 ± 0.7	87.5 ± 0.7	$\mathbf{89.9 \pm 0.3}$	80.1 ± 0.5
MRGNN	74.2 ± 1.6	86.5 ± 0.8	87.2 ± 0.9	80.6 ± 0.7
BF-GCN	77.0 ± 0.4	88.0 ± 0.4	88.1 ± 0.4	81.0 ± 0.9
BF-GraphConv	76.9 ± 1.4	$\mathbf{88.5 \pm 1.3}$	86.8 ± 0.9	$\mathbf{82.3 \pm 0.7}$
BF-MRGNN	$\mathbf{77.5 \pm 1.0}$	87.6 ± 0.6	87.9 ± 0.4	79.4 ± 0.6

The results in bold represent the best accuracy achieved for each dataset
The model selection is performed considering the results obtained on the validation set

a standard deviation apart. On WikiCS, the accuracy difference between MRGNN and its backpropagation-free version is within the standard deviation interval, with the exception of the Soft Proto version. Note that in the BF-MRGNN, due to its definition (Eq. (13)), only $\mathbf{X}$ can be used as context.

We can conclude that the proposed backpropagation-free graph convolutions are competitive with their backpropagation-based counterparts while inheriting all the advantages of backpropagation-free methods.

For what concerns the comparison between the two considered gating mechanisms (halfspace and prototype), we obtained no strong evidence in terms of accuracy in favor of using one approach over the other. However, the prototype approach does have an advantage in reducing the number of hyperparameters. In fact, it is not straightforward to define the half-space gating hyperparameters, as random initialization of hyperplanes introduces a strong assumption on the data distribution. In our experiments, we decided to keep the same parameters used in [4] for the distribution from which the weights corresponding to each hyperplane are sampled, since modifying them even slightly seemed to impact the predictive performance. On the other hand, the proposed prototype-based context function allows us to initialize the gating mechanism in a data-driven way, which turns out to be very simple since we can just uniformly sample them from the training set. The soft-gating approach shows improved performance in some cases, while in other cases, the hard-gating approach performs better. We argue that the smoothness inductive bias introduced by the soft gating makes the model less nonlinear compared to the hard-gating version. Finally, in terms of time complexity, the BackPropagation-free models present a huge advantage with respect to the standard GNN. Indeed, the computation (both forward and backward step) of each neuron is independent from all the others. Thus it is possible to perform the computation of each unit in parallel. Considering the message-passing GLNs, the layerwise construction of the model allows all neurons in the same layer to be computed in parallel. This constraint is overcome by the BL-MRGNN, where all the nodes are completely independent, and thus it is possible to parallelize the computation on all neurons.

In Table 6 we compare the performance obtained by the best configuration (selected considering the accuracy of the validation set) of the proposed BF-based models with some state-of-the-art baselines. Specifically, we consider GCN, GraphConv, GAT [37], LGC, ECG

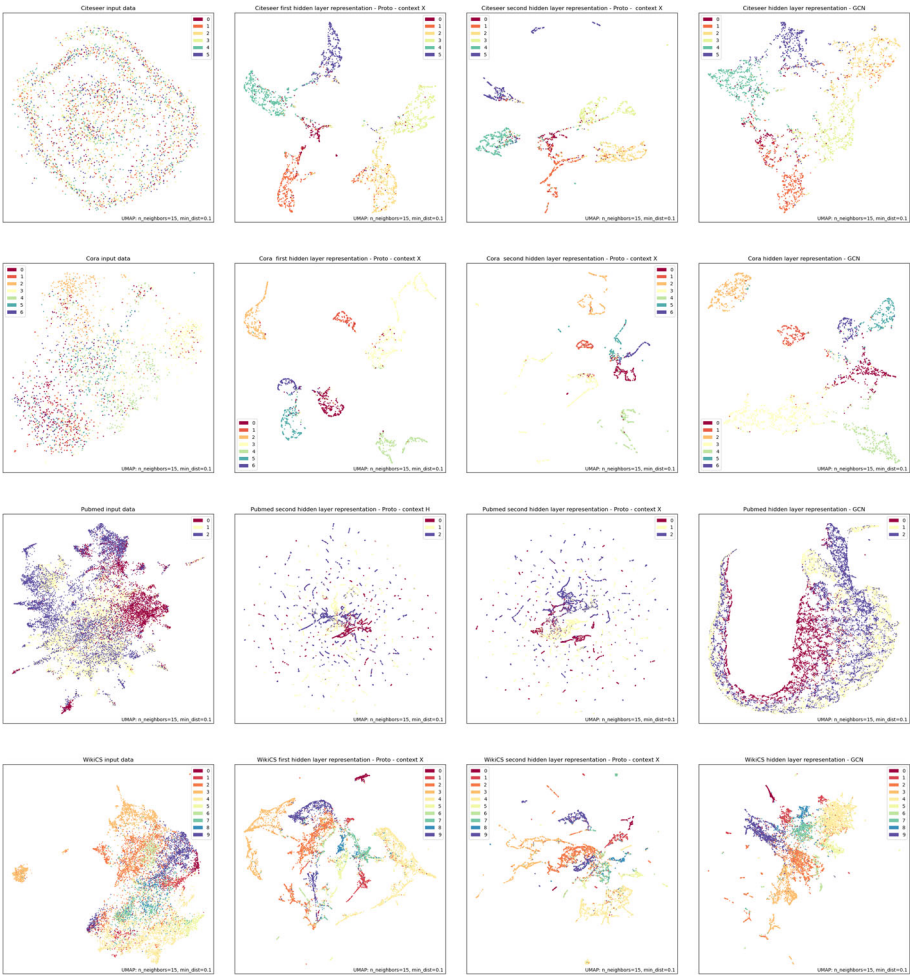


Fig. 2 UMAP 2D graphical representations of the embeddings of nodes by BF-GCN (using context X and prototypes-based gating mechanism) and GCN for all datasets. 1st column: original data; 2nd column: first layer computed by BF-GCN; 3rd column: second hidden layer computed by BF-GCN; 4nd column: last hidden last computed by the GCN. The nodes are colored based on their target class

and hLGC [38]. The comparison shows how the proposed backpropagation-free architectures achieve higher or comparable results in all the datasets. In particular, they achieved the best results on three datasets out of four (Citeseer, Cora and WikiCS) and comparable accuracy on Pubmed.

4.3 Computed representation

In Fig. 2, we used UMAP [39] to generate 2D representations of each node of the four considered datasets in the input space, its representations obtained after a single and two layers of BF-GCN using $\mathbf{X}$ as context and the representation obtained with backpropagation-based GCN. UMAP tries to maintain the distances between multi-dimensional points by

projecting them in two dimensions (allowing for human-readable data visualization). The resulting plots must be interpreted qualitatively. The network settings considered to generate these plots are selected through the validation phase (i.e., the ones reported in Tables 3, 4 and 5). The color of each node represents the class it belongs to. In Fig. 2, the first plot of each row represents the manifold of the input space. In Cora and Citeseer datasets, the positioning of the nodes is chaotic, while in Pubmed and WikiCS it is possible to note some areas where the nodes of a specific class are clustered.

Each row's second and third plots show the spatial representation obtained by the BF-GCN after the first and second hidden layers, using $\mathbf{X}$ as a context and the prototypes-based gating mechanism, respectively.

In the Cora, Citeseer and WikiCS datasets, it is possible to see that already with one hidden layer, BF-GCN representations already achieve a good class separation. At the second layer, the separation tends to increase, at least visually. Obviously, backpropagation-based GCN also achieves a good level of separation. Both with backpropagation-based GCN and the backpropagation-free counterpart, we can observe that the models tend to cluster examples belonging to the same class. In some cases, e.g., WikiCS, the resulting visualizations are pretty close, while in other cases, e.g., Pubmed or Citeseer, the distribution of the examples in the 2D space can result different. However, notice that both approaches tend to separate examples belonging to different classes. It is worth noticing that both the adopted manifold learning method (UMAP) and the hyperparameters of the model can influence the obtained plots.

4.4 BF Models and model section

One of the most costly phases of the development of a model is the selection of hyperparameters (i.e., the validation phase). Several experiments using different hyperparameters values must be run to select the model that ensures the best generalization capabilities. In our specific case, for the baseline models, we used a grid search approach, which resulted in more than 33,000 runs for each dataset (note that for each configuration, we perform five runs to have a statistically significant evaluation of the model performance). The particular structure of the BF-GNN allows us to considerably reduce the number of required runs to perform. For what concerns the GC-GCN/GraphConv, all the units that compose a layer can be trained in parallel. Thus, when adding units to the last layer, re-training the model from scratch is not required, but a pre-trained model with fewer units can be used as a starting point. Moreover, it has to be noticed that each unit (and so each layer) of the BF-GCN/GraphConv predicts the classification of the input (in a one vs rest setting). The outputs of all the neurons in a layer are then used as input for the next layer of the model. Therefore, stacking an additional layer (the l-th) does not require performing full training of a novel model, but it can be done incrementally from a previously trained model with $l-1$ layers. Clearly, this incremental validation policy significantly reduces the time required to perform the model selection phase. It is important to notice that in BF-GCN/GraphConv, the layer-wise structure poses constraints on the dependencies between the units in a layer and those in the following ones. Indeed, let us consider a network with a first layer with m_1 units and a second one with m_2 units. If we want to use a different value of units for the first layer, we have to re-train the second one from scratch. This particular constraint does not apply to the case of BF-MRGNN, where the computation of $\mathbf{R}_{l,\mathcal{T}}$ and the application of the gating neurons are entirely detached. Moreover, in BF-MRGNN, the use of the geometric mean (which does not require training) as a mixing function of the output of the various layers that computes the prediction for the

Table 7 Time comparison among the model proposed in this paper and two baselines (GCN and GraphConv)

Model	Citeseer	Cora	Pubmed	WikiCS
GCN	0.776 ± 0.189	1.610 ± 0.428	2.372 ± 1.983	1.995 ± 0.558
(l, m)	$(1, 8)$	$(2, 16)$	$(2, 48)$	$(1, 8)$
GraphConv	0.681 ± 0.152	1.602 ± 0.371	1.108 ± 0.0912	4.395 ± 0.818
(l, m)	$(2, 96)$	$(2, 128)$	$(2, 32)$	$(1, 64)$
MRGNN	0.637 ± 0.064	0.331 ± 0.0474	60.278 ± 11.352	111.465 ± 10.690
(k)	(2)	(2)	(7)	(5)
BF-GCN	0.120 ± 0.001	0.247 ± 0.001	0.233 ± 0.003	0.484 ± 0.006
Speed-up	6.3	6.5	10.2	4.1
BF-GraphConv	0.249 ± 0.001	0.246 ± 0.002	0.882 ± 0.038	0.242 ± 0.003
Speed-up	2.7	6.5	1.4	18.2
BF-MRGNN	0.038 ± 0.001	0.025 ± 0.001	0.018 ± 0.001	0.079 ± 0.0242
Speed-up	16.8	13.2	3338.8	1410.9

For each model and dataset, we report the average duration (and standard deviation) of the training phase of all five runs of the best-performing model. The time measurements are reported in seconds. Under each result of the baselines, we report the hyperparameters selected via the validation process. Under each BackPropagation-Free model, we report the speed-up obtained with respect to the same convolution optimized using the backpropagation algorithm

same class makes the addition of one or more neurons entirely transparent for the training process. Finally, the independence between the training phase of units of the same layer (and, in general, of all the units in the case of BF-MRGNN) allows for bringing down the cost of training a single model. The significant reduction of training cost can be noticed in Table 7, where we report the time required to train the best model selected through grid search. In the table, under each BF-GNN, we report the speed-up computed with respect to the corresponding baseline model, i.e., using the same convolution. In all the cases, using the Backpropagation-Free version shows a significant speed-up. Specifically, the BF-MRGNN results in a model that is always more than 10 times faster than the MRGNN reaching a speed up higher than 1000 times for Pubmed and WikiCS.

5 Conclusions and future directions

In this paper, we extended our exploration [9] of locally trainable graph convolutional operators. We presented a framework for defining backpropagation-free graph neural networks that is inspired by Gated Linear Networks. We explored variants of our approach exploiting both the common message-passing-based convolution scheme (GCN and GraphConv) and a multi-resolution graph architecture (MRGNN). The training relies on a representation space of graph nodes that is shattered into different subspaces according to the node context. Indeed, each neuron that composes the GC operator is defined as a *set* of weight vectors. A gating mechanism within each neuron selects the weight vector to use for processing the input based on its context. This mechanism allows to train each neuron independently, without using backpropagation, resulting in a set of convex problems to solve. We studied three variants of such a gating mechanism: two hard and a soft version.

We empirically assessed the performances of BF-GCN, BG-GraphConv and BF-MRGNN on four commonly adopted node classification benchmarks and verified their competitive performances. Moreover, we analyzed the behavior of such models, considering different options for shattering the context space associated with graph nodes. Finally, we evaluated the impact in terms of computational efficiency, comparing the cost in terms of computational time between the baselines and their corresponding versions using the backpropagation-free approach. The results show that the proposed BF-GNNs ensure a considerable speed-up, thanks to the possibility of parallelizing the training of the various units belonging to the same layer. Moreover, we show how the particular structure of the backpropagation-free models allows us to design a very efficient model selection strategy. We implemented layer-wise training via Stochastic Gradient Descent (SDG). Still, many other methods can be exploited to solve the resulting convex problem, making it suitable for online and continual learning scenarios. Indeed, we plan to explore the application of the proposed backpropagation-free graph neural networks to continuous learning tasks in the near future.

Author Contributions Luca Pasa and Nicolò Navarin helped in conceptualization, methodology, software, writing. Wolfgang Erb and Alessandro Sperduti were involved in conceptualization, methodology, formal analysis, writing.

Funding Open access funding provided by Università degli Studi di Padova

Declarations

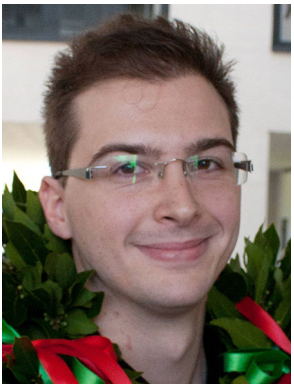
Conflict of interest The authors declare no competing interests.

References

1. Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In ICLR, citation Key: Kipf2016a. Available: arXiv:1609.02907
2. Morris C, Ritzert M, Fey M, Hamilton WL, Lenssen JE, Rattan G, Grohe M (2019) Weisfeiler and leman go neural: higher-order graph neural networks. In: Proceedings of the AAAI conference on artificial intelligence, vol 33, pp 4602–4609. Available: arXiv:1810.02244
3. Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE (2017) Neural message passing for quantum chemistry. In Proceedings of the 34th international conference on machine learning, pp 1263—-1272
4. Veness J, Lattimore T, Budden D, Bhoopchand A, Mattern C, Grabska-Barwinska A, Sezener E, Wang J, Toth P, Schmitt S et al (2021) Gated linear networks. In: Proceedings of the AAAI conference on artificial intelligence, vol 35, no 11, pp 10015–10023
5. Whittington JC, Bogacz R (2019) Theories of error back-propagation in the brain
6. Clark DG, Abbott LF, Chung S (2021) Credit assignment through broadcasting a global error vector. arXiv:2106.04089 [cs, q-bio]
7. Chen L, Chen Z, Bruna J (2020) On graph neural networks versus graph-augmented MLPs. arXiv preprint arXiv:2010.15116
8. Pasa L, Navarin N, Sperduti A (2021) Polynomial-based graph convolutional neural networks for graph classification. Mach Learn

9. Pasa L, Navarin N, Erb W, Sperduti A (2022) Backpropagation-free graph neural networks. In: IEEE international conference on data mining (ICDM), pp 388–397
10. Sperduti A, Starita A (1997) Supervised neural networks for the classification of structures. IEEE Trans Neural Netw 8(3):714–735
11. Gärtner T (2003) A survey of kernels for structured data. In: ACM SIGKDD Explorations Newsletter, 5(1), 49, citation Key: Gartner2003 publisher-place: New York, NY, USA
12. Scarselli F, Gori M, Ah Chung Tsoi AC, Hagenbuchner M, Monfardini G (2009) The graph neural network model. IEEE Trans Neural Netw 20(1):61–80
13. Micheli A (2009) Neural network for graphs: a contextual constructive approach. IEEE Trans Neural Netw 20(3):498–511
14. Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: ICLR, pp 1–14
15. Defferrard M, Bresson X, Vandergheynst P (2016) Convolutional neural networks on graphs with fast localized spectral filtering. In: NIPS, pp 3844–3852
16. Hamilton W, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: NIPS, pp 1024–1034
17. Li Y, Tarlow D, Brockschmidt M, Zemel R (2016) Gated graph sequence neural networks. In: ICLR. Available: arXiv:1511.05493
18. Xu K, Hu W, Leskovec J, Jegelka S (2019) How powerful are graph neural networks? In: International conference on learning representations
19. Tran DV, Navarin N, Sperduti A (2018) On filter size in graph convolutional networks. In: IEEE SSCI. IEEE, Bengaluru, pp 1534–1541. Available: https://ieeexplore.ieee.org/document/8628758/
20. Xinyi Z, Chen L (2019) Capsule graph neural network. In: ICLR
21. Wu F, Zhang T, de Souza AH, Fifty C, Yu T, Weinberger KQ (2019) Simplifying graph convolutional networks. In: ICML
22. Chen T, Bian S, Sun Y (2019) Are powerful graph neural nets necessary? A dissection on graph classification. arXiv preprint arXiv:1905.04579
23. Luan S, Zhao M, Chang X-W, Precup D (2019) Break the ceiling: stronger multi-scale deep graph convolutional networks. In: Advances in neural information processing systems, pp 10945–10955
24. Rossi E, Frasca F, Chamberlain B, Eynard D, Bronstein M, Monti F (2020) Sign: scalable inception graph neural networks. arXiv preprint arXiv:2004.11198
25. Pasa L, Navarin N, Sperduti A (2021) Multiresolution reservoir graph neural network. IEEE Trans Neural Netw Learn Syst 1–12
26. Veness J, Lattimore T, Budden D, Bhoopchand A, Mattern C, Grabska-Barwinska A, Sezener E, Wang J, Toth P, Schmitt S, Hutter M (2019) Gated linear networks. Available: arXiv:1910.01526
27. Veness J, Lattimore T, Bhoopchand A, Grabska-Barwinska A, Mattern C, Toth P (2017) Online learning with gated linear networks. arXiv, pp 1–40
28. Munari M, Pasa L, Zambon D, Alippi C, Navarin N (2022) Understanding catastrophic forgetting of gated linear networks in continual learning. In: 2022 International joint conference on neural networks (IJCNN), pp 1–8
29. Mattern C (2013) Linear and geometric mixtures—analysis. In: Data compression conference proceedings, pp 301–310
30. Navarin N, Erb W, Pasa L, Sperduti A (2020) Linear graph convolutional networks. In: European symposium on artificial neural networks, computational intelligence and machine learning
31. Aloupis G, Pérez-Rosés H, Pineda-Villavicencio G, Taslakian P, Trinchet D (2013) Fitting voronoi diagrams to planar tesselations. arXiv:1308.5550 [cs]
32. Cavoretto R, Rossi AD, Erb W (2021) Partition of unity methods for signal processing on graphs. J Fourier Anal Appl 27:66
33. Mernyei P, Cangea C (2020) Wiki-cs: a Wikipedia-based benchmark for graph neural networks. arXiv preprint arXiv:2007.02901
34. Fey M, Lenssen JE (2019) Fast graph representation learning with pytorch geometric. In: ICLR 2019 (RLGM Workshop)
35. Oneto L (2020) Model selection and error estimation in a nutshell. Springer, Berlin
36. Errica F, Podda M, Bacciu D, Micheli A (2020) A fair comparison of graph neural networks for graph classification. In: International conference on learning representations
37. Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2017) Graph attention networks. arXiv preprint arXiv:1710.10903
38. Pasa L, Navarin N, Erb N, Sperduti A (2023) Empowering simple graph convolutional networks. In: IEEE transactions on neural networks and learning systems, vol PP, no 99, pp 1–15

39. Narayan A, Berger B, Cho H (2020) Density-preserving data visualization unveils dynamic patterns of single-cell transcriptomic variability. bioRxiv. Available: https://www.biorxiv.org/content/early/2020/05/14/2020.05.12.077776



Luca Pasa earned his master's degree in Computer Science from the University of Padova in 2013. Subsequently, at the same institution, he successfully completed his Ph.D. in Mathematical Sciences (curriculum Computer Science) in March 2017, under the supervision of Professor Alessandro Sperduti. Between July 2017 and June 2019, he served as a postdoctoral researcher at the Center of Translational Neurophysiology Speech and Communication (CTNSC) within the Istituto Italiano di Tecnologia (IIT), working under the supervision of Dr. Leonardo Badino. Subsequently, from July 2019 to December 2021, he continued his research activity as a postdoctoral researcher at the University of Padova, affiliated with the Department of Mathematics and the Human Inspired Technologies Research Center. Since January 2022, he has held the position of Assistant Professor (RTDa) at the Department of Mathematics, University of Padova. His main research interests lie in the field of machine learning, including deep learning, computational neuroscience, and automatic speech recognition. Currently, his research activity is focused on the application of deep learning methods on structured domains (sequences, trees, and graphs). He has coauthored numerous research papers published in international refereed journals and conference proceedings. He has also served as a Program Committee member for several machine learning conferences and has actively participated in organizing various special sessions at international machine learning conferences. He is a member of the IEEE Task Force on Deep Learning and the Italian Association for Artificial Intelligence (AIxIA).

Nicoló Navarin is an associate professor in Computer Science at the Department of Mathematics "Tullio Levi-Civita", University of Padua, Italy. He got his Ph.D. in Computer Science from the University of Bologna, Italy, in 2014. He has been a visiting researcher at the University of Freiburg, Germany, at the Universitá della Svizzera Italiana, Lugano, Switzerland, and at 3IA Côte d'Azur-Interdisciplinary Institute for Artificial Intelligence, Sophia Antipolis, France. He has been a research fellow at the University of Nottingham, UK, and at the University of Padua. His research interests lie in the field of machine learning, including kernel methods and neural networks for structured data, online and continual learning, trustworthy ML, and applications to bioinformatics, business process mining, computer vision and computational psychology. At the University of Padua, he is affiliated to the Human Inspired Technology (HIT) Research Centre. He is a doctoral committee member of the PhD Program in "Brain, Mind and Computer Science". Dr. He has been serving as PC member in major machine learning conferences (ICML IJCAI, NeurIPS, ECML, AAAI, ICLR), and he has been actively involved in the organization of several special sessions (ESANN, WCCI, IJCNN) and conferences (INNS Big Data and Deep Learning 2019, International Conference on Process Mining 2020, IEEE Symposium Series in Computational Intelligence 2021, IEEE World Congress on Computational Intelligence 2022). He edited two books, and he served as guest editor for the journals Neurocomputing and IEEE Transactions on Human-Machine Systems. He is an associate editor for the journals Evolving Systems (Springer) and Neurocomputing (Elsevier), and an editorial board member for Intelligenza Artificiale (AIxIA, IOS Press). He is a member of IEEE Computational Intelligence Society, IEEE Task Force on Deep learning, IEEE Task Force on Learning from Structured Data, and of the Italian Association for Artificial Intelligence.

Wolfgang Erb is an Associate Professor in Numerical Analysis at the Department of Mathematics "Tullio Levi-Civita", University of Padua, Italy. He received his Ph.D. in Mathematics at the Technical University of Munich (Germany) in 2010. He has been a research fellow at the University of Lübeck (Germany), at the University of Eichstätt-Ingolstadt (Germany) and an Assistant Professor in Mathematics at the University of Hawaii at Manoa (US). His research interests include multivariate approximation theory, kernel methods for signal processing and learning on graphs, fast and efficient reconstruction algorithms for inverse problems, as well as applications in biomedical imaging, in particular magnetic particle imaging. He is a member of RITA (Italian Research Network on Approximation), of the Italian Mathematical Society UMI (working group TAA of approximation theory) and of GNCS-INdAM.

Alessandro Sperduti received the PhD in 1993 from University of Pisa, Italy. He is a Full Professor at the Department of Mathematics of the University of Padova. Previously, he has been Associate Professor (1998-2002) and Assistant Professor (1995-1998) at the Department of Computer Science of the University of Pisa. His research interests are mainly in neural networks, kernel methods, and process mining. He was the recipient of the